

Lucas Dalmau Garcés - Dilluns 10:30-12:30

Alvaro Enrique Martín Chango - Dimecres 10:30-12:30

Nom del projecte: **Buscamines**

## Clase Tablero (coverage 100%)

Coverage Summary for Package: model

Class	Class, %	Method, %	Line, %
Tablero	100% (1/1)	100% (9/9)	100% (52/52)

**Funcionalitat:** Inicialització de la classe Tablero

**Localització:** src/main/java/model/Tablero.java → metode desenvolupat: constructor de la classe `public Tablero(int filas, int columnas, int totalMinas, GeneradorAleatorio generadorAleatorio)`

**Test:** src/main/test/model/TableroTest.java → `void iniciarClaseTableroTest()`

test de caixa negra on es comproven els valors limit i els valors interiors i exteriors a la frontera amb una partició equivalent de valors vàlids/invalids.

**Comentari:** Gràcies a aquest primer test aconseguim statment coverage i path coverage per al constructor `Tablero(int filas, int columnas, int totalMinas)` y per el mètode `inicializarMatrizTablero()`

```
public Tablero(int filas, int columnas, int totalMinas, GeneradorAleatorio generadorAleatorio) {
    //aplicamos diseño por contrato
    // las precondiciones de inicializar un tablero es que los parametros que le pasen sean validos
    assert filas > 0 : "El número de filas debe ser mayor que 0";
    assert columnas > 0 : "El número de columnas debe ser mayor que 0";
    assert totalMinas >= 0 : "El número de minas debe ser 0 o mayor que 0";
    assert totalMinas <= (columnas * filas) : "No se pueden poner mas minas que casillas tenga el tablero";

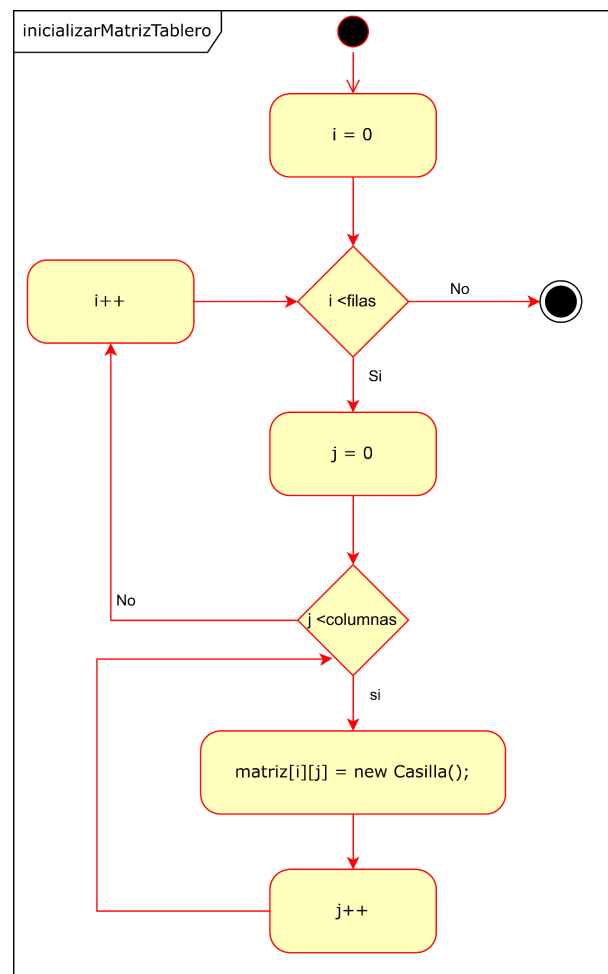
    this.filas = filas;
    this.columnas = columnas;
    this.totalMinas = totalMinas;
    this.matriz = new Casilla [filas][columnas];
    this.generadorAleatorio = generadorAleatorio;

    inicializarMatrizTablero(); //pone las casillas dentro de cada celda de la matriz
    ponerMinas(); //ponemos las minas de forma aleatoria

    //postcondiciones --> que se haya inicilaizado con el total de minas correspondientes
    int minasPuestas = 0;
    for(int i=0; i < filas; i++){
        for(int j=0; j < columnas; j++){
            if(matriz[i][j].getTieneMina()){
                minasPuestas++;
            }
        }
    }
    assert(totalMinas == minasPuestas); // hay que vigilar que tengamos todas las minas

    // contamos las minas que hay al lado de una casilla vacia para que cuando esta se muestre,
    // ensñe el numero de minas que tiene alrededor
    calcularMinasAdyacentes();
}

private void inicializarMatrizTablero() {
    // doble bucle que recorra la matriz haci
    for (int i= 0; i<filas; i++){
        for (int j=0; j<columnas; j++){
            matriz[i][j] = new Casilla();
        }
    }
}
```



**Funcionalitat:** col·locació de mines de forma aleatòria dintre de la matriu de caselles

**Localització:** src/main/java/model/Tablero.java → mètode desenvolupat: `private void ponerMinas()`

**Test1:** src/main/test/model/TableroTest.java → `void ponerMinasTest()` test de caixa negra per provar els valors límit i els valors interiors i exteriors a la frontera amb una partició equivalent de valors vàlids/invalids utilitzant a més un mock object que encapsula una funció de aleatorització de forma que els valors 'aleatoris' s'introdueixen en el mock object com un array de numeros on el mètode ponerMinas() indirectament extreu el següent valor del array permetent col·locar les mines en les posicions que nosaltres volem per a poder fer el test

**Test2:** src/main/test/model/TableroTest.java → `void loopTestingPonerMinas()`

Test específicament fet per a fer un loop testing del mètode `ponerMinas()`. Aquest test analitza els següents casos sobre una matriu de caselles 3 \* 3: saltar el loop, fer una passada, fer dues passades, fer 4 passades (n passades per el interior), fer 8 passades (límit interior), fer 9 passades (valor frontera).

```
private void ponerMinas() {
    // selecciona sitios aleatorios donde poner las minas (pone a true
    // donde ponga la mina a las de al lado hay que incrementa el co

    //OBS hay que tener cuidado de que si se pone la mina en un limit

    int minasPuestas =0;
    int MAX_minas = getFilas() * getColumnas();

    while (minasPuestas < totalMinas && minasPuestas < MAX_minas ) {
        int fila = generadorAleatorio.obtenerAleatorio(filas);
        int columna = generadorAleatorio.obtenerAleatorio(columnas);

        if(!matriz[fila][columna].getTieneMina()){ //si no habiamos col
            matriz[fila][columna].setTieneMina(true);
            minasPuestas++;
        }
    }
}
```

**Funcionalitat:** Càlcul de mines adjacents a les caselles buides

**Localització:** src/main/java/model/Tablero.java → mètode desenvolupat: `private void calcularMinasAdyacentes()`

**Test:**src/main/test/model/TableroTest.java→`void calcularMinasAdyacentesTest()`

test de caixa negra on es comproven els valors límit i els valors interiors i exteriors a la frontera amb una partició equivalent de valors vàlids/invalids. A mes, aquest test aconsegueix statement coverage i path coverage per el propi metode avaluat.

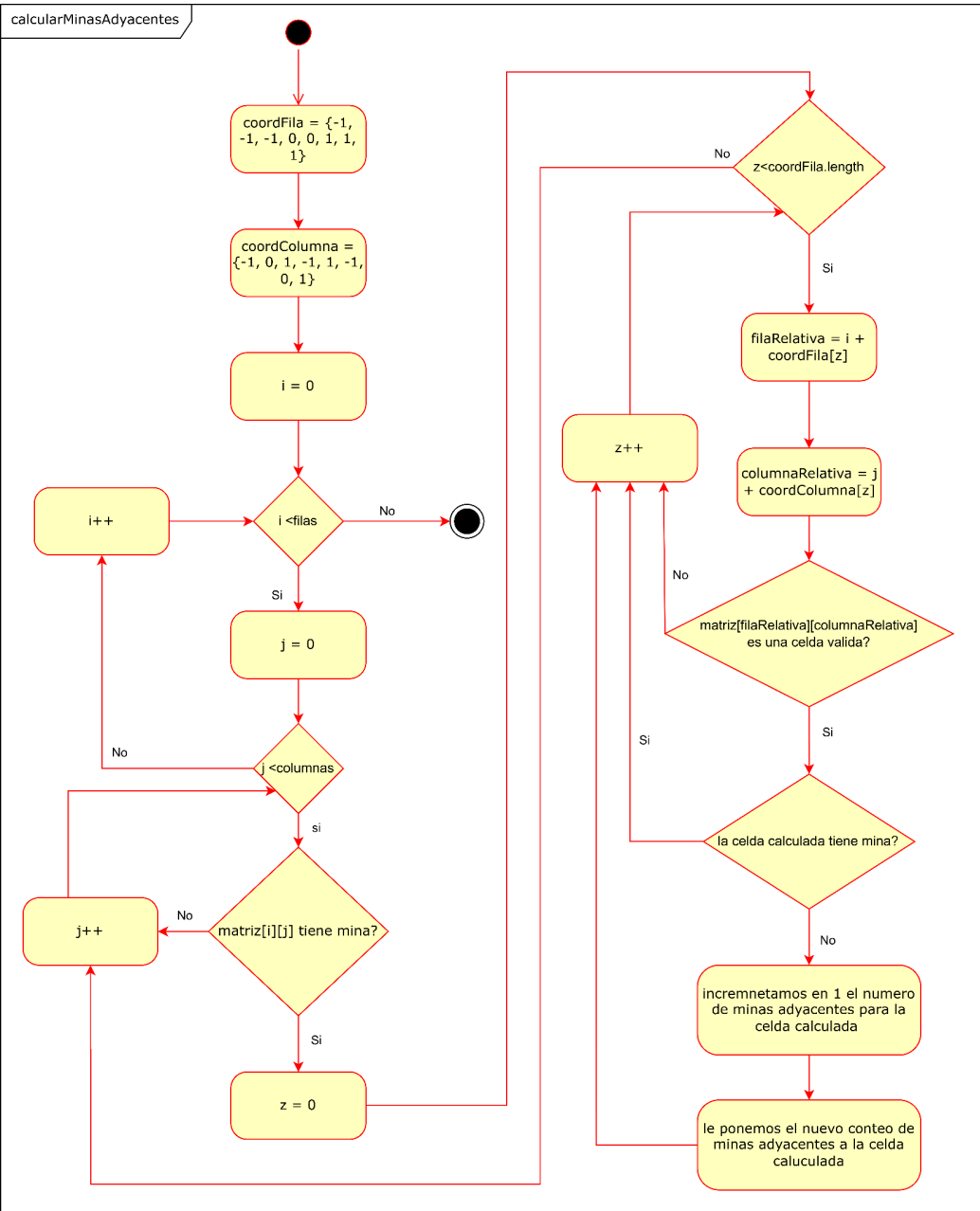
```
private void calcularMinasAdyacentes() {

    int[] coordFila = {-1, -1, -1, 0, 0, 1, 1, 1};
    int[] coordColumna = {-1, 0, 1, -1, 1, -1, 0, 1};

    for (int i= 0; i<filas; i++){
        for (int j=0; j<columnas; j++){
            if ( matriz[i][j].getTieneMina() ){ // si encontramos una mina, buscamos sus casillas adyacentes
                for (int z=0; z<coordFila.length; z++){

                    //calculamos la posicion a mirar
                    int filaRelativa = i + coordFila[z];
                    int columnaRelativa = j + coordColumna[z];

                    if (celdaValida(filaRelativa,columnaRelativa)) // siempre y cuando la posicion sea valida
                        if (!matriz[filaRelativa][columnaRelativa].getTieneMina()) { // y no sean una mina tambien
                            //sumamos 1
                            int minasActuales = matriz[filaRelativa][columnaRelativa].getMinasAdyacentes() + 1;
                            matriz[filaRelativa][columnaRelativa].setMinasAdyacentes(minasActuales);
                        }
                    }
                }
            }
        }
    }
}
```



**Funcionalitat:** Calcular si donada una fila i una columna, la casella per aquest valors es vàlida (es troba dintre del tauler)

**Localització:** src/main/java/model/Tablero.java → mètode desenvolupat: `public boolean celdaValida(int fila, int columna)`

**Test1:** src/main/test/model/TableroTest.java → `void celdaValidaTest ()`

test de caixa negra on es comproven els valors límit i els valors interiors i exteriors a la frontera amb una partició equivalent de valors vàlids/invalids.

**Test2:** src/main/test/model/TableroTest.java → `void conditionCoverageCeldaValidaTest ()`

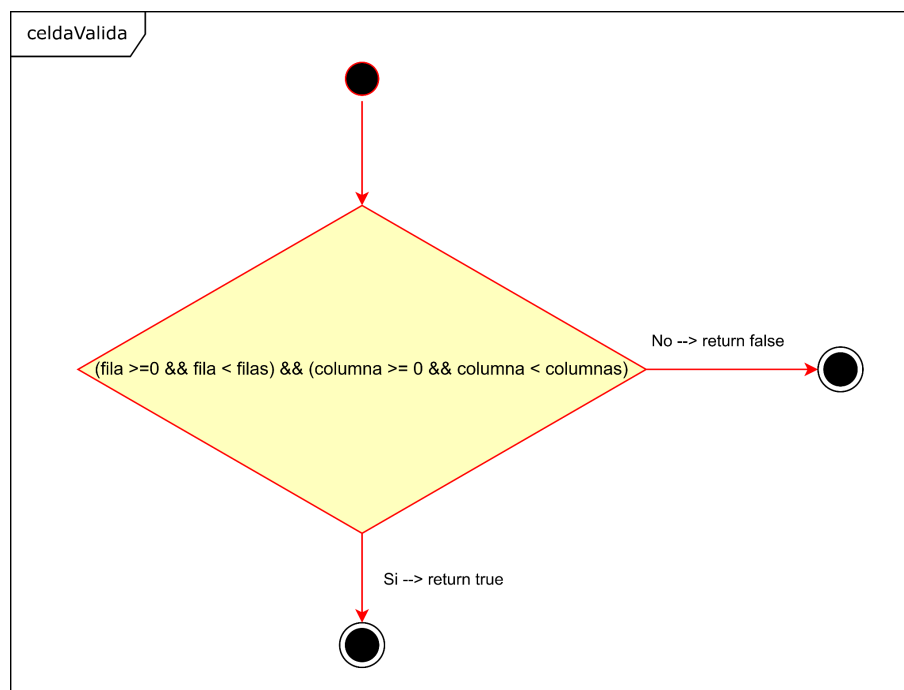
Aquest test ens assegura que cada condició de l'única expressió lògica que conté el mètode: `(fila >=0 && fila < filas) && (columna >= 0 && columna < columnas)` prenen el valor de true i false almenys una vegada

**Test3:** src/main/test/model/TableroTest.java → `void decisionCoverageCeldaValidaTest ()`

Aquest test ens assegura que la decisió `(fila >=0 && fila < filas) && (columna >= 0 && columna < columnas)` pren el valor de true i false almenys una vegada

**Comentari:** gracies a aquests test també hem complert statement coverage i path coverage

```
public boolean celdaValida(int fila, int columna){  
    return (fila >=0 && fila < filas) && (columna >= 0 && columna < columnas);  
}
```



## Clase GestorDelJuego(coverage 100%)

### Coverage Breakdown

Package	Class, %	Method, %	Line, %
controlador	100% (1/1)	100% (16/16)	100% (88/88)

**Funcionalitat:** Mètode que permet processar els moviments que vol fer l'usuari, agafant la celda on vol fer l'acció i la jugada ( 1 → Revelar celda , 2 → Posar Bandera, 3 → Treure Bandera.

Retorna true si s'ha pogut realitzar el moviment i false si no s'ha pogut.

**Localització:** src/main/java/controlador/GestorDelJuego.java → mètode desenvolupat:

```
public void realitzar_jugada(int fila, int col, int jugada)
```

### Test:

Hem dividit aquest test en diversos tests mes petits per tal de facilitar el treball i perquè sigui més ordenat → Com hi ha diversos moviments hem de tenir en compte cadascun i veure que es facin bé. Arxiu: src/main/test/java/controlador/GestorDelJuegoTest.java

**Test1:** `void revelarMovimientoTest()` Test de caixa negra que prova que el moviment de revelar celda funciona correctament. Aquests test cobreix els casos on es posa com a jugada revelar casella, i es comproven els límits interiors i exteriors del taulell per tal de que no es pugui realitzar la jugada fora del taulell mitjançant la tècnica de partició equivalent i valors límit. També es fan comprovacions per veure que no es revelin caselles que ja estan revelades.

**Test2:** `void flagMovimientoTest()` Test de caixa negra que verifica el moviment de col·locar banderes on es tracta de posar una bandera en una casella no revelada, en una casella no valida, en una casella revelada i en una casella que ja tingui una bandera (mitjançant la tècnica de partició equivalent i valors límit).

**Test3:** `void quitarBanderaTest()` test de caixa negra que implementa els mateixos casos de prova mencionats al punt anterior però tractant de treure la bandera en comptes de posar-la. Tot això utilitzant la tècnica de partició equivalent .

**Test4:** `void movimientoNoValidoTest()` test de caixa negra per assegurar-se que les jugades no vàlides són rebutjades correctament.

**Comentari:** gràcies a tots aquest tests obtenim statement coverage d'aquest mètode

```
153
154 public boolean realitzar_jugada(int fila, int col, int jugada){
155     if(fila < 0 || fila >= tablero.getFilas() || col < 0 || col >= tablero.getColumns())
156         return false;
157     else{
158         switch (jugada){
159             case 1:
160                 return revelarCelda(fila,col);
161             case 2:
162                 return flagCelda(fila,col);
163             case 3:
164                 return removeBandera(fila,col);
165             default:
166                 return false;
167         }
168     }
169 }
170 }
```

**Funcionalitat:** Configurar el joc amb dimensions del tauler i nombre de mines introduïdes per l'usuari

**Localització:** src/main/java/controlador/GestorDelJuego.java → Mètode desenvolupat:  
`public void configurarJuego()`

**Test:** Arxiu → src/main/test/java/controlador/GestorDelJuegoTest.java

**Test1:** `void configuracionJuegotest()`

Test de caixa negra que prova el flux de configuració del joc amb valors vàlids per al nombre de files, columnes i mines. Utilitza mock de l'entrada per simular una configuració correcta.

**Test2:** → `void inputFueraRangoTest()`

Test de caixa negra que verifica el comportament quan l'usuari introdueix valors fora de rang per a files, columnes o mines utilitzant particions equivalents de inputs vàlids i invalids.

**Test3:** `void inputNoValidoConfiguracionTest()` Test de caixa negra que comprova que les entrades que no siguin numeriques generin conflictes generant una excepció i impedit la configuració errònea del joc.

**Comentari:** gràcies a tots aquest tests obtenim statement coverage d'aquest mètode.

```
public void configurarJuego(){
    Scanner scanner = new Scanner(System.in);
    int filas = 0, columnas = 0, minas = 0;

    interfaz.mostrarMensaje("Bienvenido al Buscaminas :)");
    interfaz.mostrarMensaje("Vamos a configurar el tablero.");

    while (true) { // Bucle que pide configuración del tablero hasta que sea válida.
        try {
            interfaz.mostrarMensaje("Introduce el número de filas (1-7):");
            filas = Integer.parseInt(scanner.nextLine());

            interfaz.mostrarMensaje("Introduce el número de columnas (1-7):");
            columnas = Integer.parseInt(scanner.nextLine());

            interfaz.mostrarMensaje("Introduce el número de minas (1-" + (filas * columnas) + "):");
            minas = Integer.parseInt(scanner.nextLine());
            //Comprueba que los datos introducidos sean válidos
            if (filas > 0 && filas <= 7 && columnas > 0 && columnas <= 7
                && minas > 0 && minas <= filas * columnas) {
                break;
            } else {
                interfaz.mostrarMensaje("Error: filas, columnas y minas deben estar entre 1 y 7.");
                interfaz.mostrarMensaje("Además, las minas no pueden superar filas x columnas.");
            }
        } catch (NumberFormatException e) {
            interfaz.mostrarMensaje("Error: Introduzca solo números válidos.");
        }
    }

    // Inicializar el tablero con las dimensiones y minas proporcionadas.
    this.setTablero(new Tablero(filas, columnas, minas, new GeneradorAleatorioDefault(new Random())));
    this.setCasillasRestantes(filas*columnas);
    interfaz.mostrarMensaje("¡Tablero configurado exitosamente con " + filas + " filas, " + columnas + " columnas y " + minas + " minas!");
}
```

**Funcionalitat:** Revelar una casella del tauler

**Localització:** src/main/java/controlador/GestorDelJuego.java → Mètode desenvolupat:  
`public boolean revelarCelda(int fila, int col)`

**Test:** Arxiu: src/main/test/java/controlador/GestorDelJuegoTest.java→

**Test1:** `void revelarMovimientoTest()`

Test de caixa negra que valida el comportament del mètode a través de l'utilització de valors límit i els valors interiors a la frontera y exteriors a la frontera. aquest test també ens assegura que el test també ens assegura que el joc es comporta correctament quan es revela una mina s'acaba la partida, així com en casos de jugades vàlides i no vàlides utilitzant les particions equivalents. A més, gràcies a aquest obtenim statement, decision i condition coverage .

**Test2:** `void detectarMinaTest()`

Test de caixa negra que valida el comportament del mètode al revelar una casella que conté una mina. Aquest test ens assegura que quan es reveli una mina, el joc s'acabi i hagi perdut. Permet fer el statement coverage de la funció i juntament amb el test anterior es compleix el decision i condition coverage.

```
public boolean revelarCelda(int fila, int col){  
  
    if(tablero.getCasilla(fila,col).getRevelada())  
        return false;  
  
    if(tablero.getCasilla(fila,col).getTieneMina()){  
        System.out.println("BOOM ha estallado una mina, has perdido.");  
        this.setFinalJuego(true);  
        return true;  
    }  
    tablero.getCasilla(fila,col).setRevelada(true);  
    this.setCasillasRestantes(this.getCasillasRestantes() - 1);  
    return true;  
}
```

**Funcionalitat:** Col·locar una bandera en una casella

**Localització:** src/main/java/controlador/GestorDelJuego.java → Mètode desenvolupat:  
`public boolean flagCelda(int fila, int col)`

**Test:** src/main/test/java/controlador/GestorDelJuegoTest.java

`void flagMovimientoTest()`

Test de caixa negra que prova valors dins del tauler i valors de fora d'aquest. També testeja els valors no vàlids amb particions equivalents, on es tracta de posar una bandera en una casella ja revelada o una casella que ja té una bandera d'abans. A més, aquest test ens proporciona condition coverage ja que es testeja totes les condicions que formen part de l'expressió condicional tant a true i com a false almenys una vegada. També aconseguim decision coverage ja que testeja una jugada vàlida que fa que la condició s'avalui com a true i una no vàlida que fa que s'avalui la condició com a false. Com es pot apreciar a la captura següent, també tenim statement coverage.

```

public boolean flagCelda(int fila, int col){

    if(tablero.getCasilla(fila,col).getRevelada() || tablero.getCasilla(fila,col).getTieneBandera())
        return false;
    else{
        tablero.getCasilla(fila,col).setTieneBandera(true);
        //Si ponemos una flag --> Se quita esa casilla de casillas restantes si realmente tiene mina
        if(this.getTablero().getCasilla(fila,col).getTieneMina()){
            this.setCasillasRestantes(this.getCasillasRestantes() - 1);
        }
        return true;
    }
}

```

**Funcionalitat:** Treure una bandera d'una casella

**Localització:** src/main/java/controlador/GestorDelJuego.java → Mètode desenvolupat:  
**public boolean removeBandera(int fila, int col)**

**Test:** src/main/test/java/controlador/GestorDelJuegoTest.java

**void quitarBanderaTest()**

Test de caixa negra que utilitza tècniques com particions equivalents, i condicions límit per testejar la correcta retirada de la bandera en caselles vàlides i no vàlides utilitzant valors límit, valors interiors a la frontera i valors exteriors a la frontera . A més, obtenim statement coverage tal i com es comprova a la següent captura de pantalla:

```

public boolean removeBandera(int fila, int col) {

    if (!tablero.getCasilla(fila, col).getTieneBandera()) {
        return false;
    } else {
        tablero.getCasilla(fila, col).setTieneBandera(false);
        //si quitamos la bandera y tenia mina --> Se suma una a las casillas restantes -->
        if (tablero.getCasilla(fila, col).getTieneMina())
            this.setCasillasRestantes(this.getCasillasRestantes() + 1);
        return true;
    }
}
}

```

**Funcionalitat:** Controlar el flux del Joc.

**Localització:** src/main/java/controlador/GestorDelJuego.java → Mètode desenvolupat:  
**public boolean empezarJuego()**

**Test:** src/main/test/java/controlador/GestorDelJuegoTest.java → Hem dividit els casos en 6 tests per tal de cobrir el màxim nombre de casos i poder fer el statement coverage.

**Test 1 → void partidaConMinaTest()**

Test de caixa negra que et comprova com reacciona el joc davant el descobriment d'una mina. Aquest test ens assegura que quan es revela una mina el joc s'acabi, es doni com perdut i surti un missatge dient que el jugador ha perdut.

En el test es simula l'entrada d'una jugada de l'usuari i la reacció del joc davant l'entrada.

Aquest test juntament amb els altres 5 fan statement coverage de la funció.

**Test 2 → void ganarPartidaBanderaTest()**

Test de caixa negra que comprova com reacciona el joc davant la jugada de posar la bandera, quan hauria de guanyar. Es comprova que la variable que indica si el jugador ha guanyat estigui a true i permet veure com el gestor del joc es desenvolupa amb aquest cas. Tot això simulant l'entrada d'una jugada per part de l'usuari.

Aquest test juntament amb els altres 5 fan statement coverage de la funció.

**Test 3 → void ganarPartidaRevealTest()**



Test de caixa negra que comprova com reacciona el joc davant la jugada de revelar l'última casella sense mina (el resultat hauria de ser victòria del jugador). Es comprova que la variable que indica si el jugador ha guanyat estigui a true i permet veure com el gestor del joc es desenvolupa amb aquest cas. Tot això simulant l'entrada d'una jugada per part de l'usuari.

Aquest test juntament amb els altres 5 fan statement coverage de la funció.

**Test 4** → `void partidaInputNoValidoTest()`

Test de caixa negra que comprova que passa si es passa un input no vàlid al joc, és a dir, el joc funciona amb l'entrada de números. Amb aquest test el que comprovem és que passa si es passa un caràcter que no sigui de tipus int i.

En aquest test fem ús de mock objects fets per mockito on simulen la interfície i utilitzem un verify per veure si es mostra per pantalla el missatge correcte per el tipus d'error.

Aquest test juntament amb els altres 5 fan statement coverage de la funció.

**Test 5** → `void partidaMalFormatoTest()`

Test de caixa negra que comprova que passa si es passa un input amb un format no vàlid, és a dir, el joc funciona amb l'entrada de 3 números, però amb aquest test el que comprovem és com hauria de reaccionar el joc quan no es passen aquests tres paràmetres (si es passen mes o menys) → partició equivalent amb paràmetres invalids .

En aquest test fem ús de mock objects fets per mockito on simulen la interface i utilitzem un verify per veure si es mostra per pantalla el missatge correcte per el tipus d'error.

Aquest test juntament amb els altres 5 fan statement coverage de la funció.

**Test 6** → `void partidaMalFormato1Test()`

Test de caixa negra que comprova que passa si es passa un input amb un format no vàlid, és a dir, el joc funciona amb l'entrada de 3 números, però amb aquest test el que comprovem és com hauria de reaccionar el joc quan no es passen aquests tres paràmetres (si es passen mes o menys) → partició equivalent amb paràmetres invalids .

En aquest test fem ús de mock objects fets per mockito on simulen la interface i utilitzem un verify per veure si es mostra per pantalla el missatge correcte per el tipus d'error.

Aquest test juntament amb els altres 5 fan statement coverage de la funció.

```
118 public void empezarJuego() {
119     Scanner scanner = new Scanner(System.in);
120     interfaz.mostrarMensaje("Introduce tu movimiento en el formato: fila columna acción (1=Revelar, 2=Poner bandera, 3=Quitar bandera)");
121
122     while (!finalJuego) {
123         interfaz.mostrarTablero(tablero);
124
125         String input = scanner.nextLine();
126         String[] parts = input.split(" ");
127
128         //Comprobar datos de entrada + su tratamiento
129         if (parts.length == 3) {
130             try {
131                 int fila = Integer.parseInt(parts[0]);
132                 int col = Integer.parseInt(parts[1]);
133                 int action = Integer.parseInt(parts[2]);
134
135                 if (realizar_jugada(fila, col, action)) {
136                     if (casillasRestantes <= tablero.getTotalMinas() && !finalJuego) {
137                         interfaz.mostrarMensaje("¡Felicidades, has ganado!");
138                         hasGanado = true;
139                         finalJuego = true;
140                         return;
141                     }
142                 }
143             } catch (NumberFormatException e) {
144                 interfaz.mostrarMensaje("Error: Introduce números válidos.");
145             }
146         } else {
147             interfaz.mostrarMensaje("Formato incorrecto. Usa: fila columna acción.");
148         }
149     }
150 }
151 }
```

